Introduction to Data Management
CSE 344

Lecture 6: Nested Queries in SQL
Announcements

• Webquiz 2 is due on Tuesday
  – Webquiz 3 will be out tomorrow evening

• Homework 2 is due on Wednesday
Lecture Goals

• Today we will learn how to write (even) more powerful SQL queries

• Reading: Ch. 6.3
Subqueries

• A subquery is a SQL query nested inside a larger query
• Such inner-outer queries are called nested queries
• A subquery may occur in:
  – A SELECT clause
  – A FROM clause
  – A WHERE clause

• Rule of thumb: avoid writing nested queries when possible
  – But sometimes it’s impossible, as we will see
Subqueries...

- Can appear as computed values in a SELECT clause.
- Can appear in FROM clauses and aliased using a tuple variable that represents the tuples in the result of the subquery.
- Can return a single constant to be compared with another value in a WHERE clause.
- Can return relations to be used in WHERE clauses.
1. Subqueries in SELECT

Product (pname, price, cid)
Company (cid, cname, city)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city FROM Company Y WHERE Y.cid=X.cid) as City
FROM Product X
```

What happens if the subquery returns more than one city? We get a runtime error (and SQLite simply ignores the extra values...)

“correlated subquery”
1. Subqueries in SELECT

Whenever possible, don’t use a nested queries:

```sql
SELECT X.pname, (SELECT Y.city
FROM Company Y
WHERE Y.cid=X.cid) as City
FROM Product X
```

We have “unnested” the query.
1. Subqueries in SELECT

Compute the number of products made by each company

```sql
SELECT DISTINCT C.cname, (SELECT count(*)
FROM Product P
WHERE P.cid=C.cid)
FROM Company C
```
Product (pname, price, cid)
Company (cid, cname, city)

1. Subqueries in SELECT

Compute the number of products made by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
                  FROM Product P
                  WHERE P.cid=C.cid)
FROM Company C
```

Better: we can unnest using a GROUP BY

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```
1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
  FROM Product P
  WHERE P.cid=C.cid)
FROM Company C
```

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```
1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
FROM Product P
WHERE P.cid=C.cid)
FROM Company C
```

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```

No! Different results if a company has no products

```
SELECT C.cname, count(pname)
FROM Company C LEFT OUTER JOIN Product P
ON C.cid=P.cid
GROUP BY C.cname
```
2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

```
SELECT X.pname
FROM (SELECT *
    FROM Product AS Y
    WHERE price > 20) as X
WHERE X.price < 500
```

Try unnest this query!
2. Subqueries in FROM

At the end of the lecture we will see that sometimes we really need a subquery and one option will be to put it in the FROM clause.
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Product (pname, price, cid)
Company (cid, cname, city)
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using **EXISTS**:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT *
               FROM Product P
               WHERE C.cid = P.cid AND P.price < 200)
```
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using IN

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price < 200)
```
3. Subqueries in WHERE

Using **ANY**:  

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
                  FROM Product P
                  WHERE P.cid = C.cid)
```

Find all companies that make some products with price < 200
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using **ANY**:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
                    FROM Product P
                    WHERE P.cid = C.cid)
```

Existential quantifiers

Not supported in sqlite
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Now let’s unnest it:

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Now let’s unnest it:

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

1. Find the other companies that make some product ≥ 200

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                FROM Product P
                WHERE P.price >= 200)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

1. Find the other companies that make some product ≥ 200

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
    FROM Product P
    WHERE P.price >= 200)
```

2. Find all companies s.t. all their products have price < 200

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid NOT IN (SELECT P.cid
    FROM Product P
    WHERE P.price >= 200)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
                 FROM Product P
                 WHERE P.cid = C.cid AND P.price >= 200)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Using **ALL**:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL (SELECT price
 FROM Product P
 WHERE P.cid = C.cid)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Using **ALL**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL (SELECT price
FROM Product P
WHERE P.cid = C.cid)
```

Not supported in sqlite
Question for Database Fans and their Friends

• Can we unnest the *universal quantifier* query?

• We need to first discuss the concept of *monotonicity*
Monotone Queries

• Definition A query Q is monotone if:
  – Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.
Monotone Queries

- Definition A query Q is **monotone** if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td>cid</td>
</tr>
<tr>
<td>price</td>
<td>cname</td>
</tr>
<tr>
<td>Gizmo</td>
<td>c002</td>
</tr>
<tr>
<td>Gadget</td>
<td>c001</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
</tbody>
</table>

CSE 344 - Fall 2016
Monotone Queries

- Definition: A query \( Q \) is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.
Monotone Queries

• **Theorem**: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.
Monotone Queries

• **Theorem:** If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

• **Proof.** We use the nested loop semantics: if we insert a tuple in a relation $R_i$, this will not remove any tuples from the answer.
Monotone Queries

• The query:

Find all companies s.t. all their products have price < 200

is not monotone
Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200

is not monotone

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c001</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunworks</td>
</tr>
</tbody>
</table>
Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200

is not monotone

- Consequence: we cannot write it as a SELECT-FROM-WHERE query without nested subqueries
Queries that must be nested

• Queries with universal quantifiers or with negation
Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that use aggregates in certain ways
  - \texttt{sum(..)} and \texttt{count(*)} are NOT monotone, because they do not satisfy set containment
  - \texttt{select count(*) from R} is not monotone!